| | | |
|---|---|---|
| DISVE<br>Pôle Licence | **ANNEE UNIVERSITAIRE 2009/2010**<br>**SESSION 1 D'AUTOMNE**<br><br>**PARCOURS / ETAPE :** CSB3, MHT53 **Code UE :** INF 251<br>**Epreuve :** Algorithmes et Structures de Données Fondamentaux<br>**Date :** 10 Juin 2010 **Heure :** 8H30 **Durée :** 1H30<br>Documents : non autorisés<br>Epreuve de M/Mme : DELEST<br>**Vous devez répondre directement sur le sujet qui comporte 8 pages.**<br>**Insérez ensuite votre réponse dans une copie d'examen comportant tous les renseignements administratifs** | |

Indiquez votre code **d'anonymat** : N° : .

**La notation tiendra compte de la clarté de l'écriture des réponses.**

Barème indicatif

- Question 1 – Connaissances générales : 2 points
- Question 2 – Arbres binaires de recherche : 6 points
- Question 3 – Connaissance des structures de données : 6 Points
- Question 4 – Utilisation des structures de données : 6 points

**Question 1.** Cochez les affirmations correctes et quelle que soit la réponse justifiez sur la ligne en pointillé.

☐ Dans un arbre binaire de recherche le maximum est toujours placé sur une feuille.

……

☐ Un arbre AVL de taille 15 n'est jamais un arbre binaire de recherche.

……

☐ Dans un B-arbre les valeurs se trouvent uniquement sur les feuilles.

……

☐ Si s est une structure dont un des champs x est un pointeur d'entier, on accède à l'entier par s.x^.

……

**Question 2.** Soit la liste de clé A=(8,1,4,3,7,6,5,2).

1 – Ecrire la fonction *supprimer* qui supprime un élément dans un arbre binaire de recherche.

2 - Construire l'arbre binaire de recherche B correspondant à l'insertion consécutive des clés de la liste A. On dessinera l'arbre après chacune des quatre premières insertions ainsi que l'arbre final.

3 – Donnez pour l'arbre B la liste des sommets en ordre préfixe, en ordre infixe et en ordre suffixe.

4 - Montrez l'exécution de la suppression de la clé 4 sur l'arbre B.

5 – Construire l'AVL correspondant à l'insertion consécutive de la liste A. On dessinera l'arbre après chacune des quatre premières insertions ainsi que l'arbre final.

6 – Construire le tas Max correspondant à l'insertion consécutive de la liste A. On dessinera l'arbre final.

**Question 3.** Soit une suite de clés de l'ensemble {0,1} stockées dans une liste L. Par exemple L=(0, 0,1,0,0,1,1,1).

1 – Ecrire la fonction booléenne *verifier* qui retourne vrai quand dans la liste L, il y a autant de valeurs 0 que de valeurs 1. Elle retourne faux dans le cas contraire. On utilisera les primitives du type abstrait listeSC.

2 – On souhaite disposer d'une fonction *complete* qui complète la liste L lorsque la fonction *vérifier* retourne faux. Pourquoi est-il plus facile de réaliser cette fonction si la liste est doublement chainée que si elle est simplement chainée ?

3 – Ecrire la fonction *complete* en utilisant le type abstrait listeDC.

4 – Ecrire cette même fonction *complete* en utilisant une implémentation dynamique à la place des primitives.

5 – Ecrire la fonction *separe* qui à partir d'une liste doublement chainée fournit 2 structures :

- une pile contenant autant de 0 que ceux contenus dans la liste,
- une file contenant autant de 1 que ceux contenus dans la liste.

**Question 4.** Un magasin a une seule porte d'entrée, il y a plusieurs vendeurs et une seule caisse pour payer. Les clients rentrent tous par la porte d'entrée, ils attendent pour être pris en charge par un vendeur, puis, après avoir été servis, ils vont à la caisse où ils attendent leur tour pour payer. Parfois un client excédé par l'attente quitte le magasin.

1 – Quelle structure de donnée permet de modéliser la plupart des éléments ci-dessus ? Pourquoi ?

2 – Ecrivez les structures de données client, vendeur, caisse, magasin.

3 - Décrivez le(s) type(s) abstrait(s) correspondant à ces structures et tout particulièrement les primitives.

4 - Ecrire les primitives d'accès et de modification du type vendeur ?

5 – Si ce magasin est muni d'un système de carte de fidélité, une fonctionnalité de la caisse consiste à proposer la saisie du nom du client. Quelle structure de donnée peut permettre de gérer cette fonction ? Pourquoi ? Modifier le type abstrait de la question 3 en conséquence. On ne demande pas d'écrire les primitives.

**Listes simplement chainées (listeSC)**

fonction valeur(val L:liste d'objet):objet;
fonction debutListe(val L:liste d'objet) :vide ;
fonction suivant(val L:liste d'objet) :vide ;
fonction listeVide(val L:liste d'objet): booleen;
fonction créerListe(ref L:liste d'objet):vide;
fonction insérerAprès(ref L:liste d'objet; val x:objet;):vide;
fonction insérerEnTete(ref L:liste d'objet  val x:objet):vide;
fonction supprimerAprès(ref L:liste d'objet):vide;
fonction supprimerEnTete(ref L:liste d'objet):vide;

**Listes doublement chainées (listeDC)**

fonction finListe(val L:liste d'objet):vide;
fonction précédent(val L::liste d'objet): vide;

**Piles**

fonction valeur(ref P:pile de objet):objet;
fonction fileVide(ref P:pile de objet):booléen;
fonction créerPile(P:pile de objet) :vide
fonction empiler(ref P:pile de objet;val x:objet):vide;
fonction dépiler(ref P:pile de objet):vide;
fonction detruirePile(ref P:pile de objet):vide;

**Files**

fonction valeur(ref F:file de objet):objet;
fonction fileVide(ref F:file de objet):booléen;
fonction créerFile(F:file de objet);vide;
fonction enfiler(ref F:file de objet;val x:objet):vide;
fonction défiler (ref F:file de objet):vide;
fonction detruireFile(ref F:file de objet):vide;

**Arbres binaires**

fonction getValeur(val S:sommet):objet;
fonction filsGauche(val S:sommet):sommet;
fonction filsDroit(val S:sommet):sommet;
fonction pere(val S:sommet):sommet;
fonction setValeur(ref S:sommet;val x:objet):vide;
fonction ajouterFilsGauche(ref S:sommet,val x:objet):vide;
fonction ajouterFilsDroit(ref S:sommet,x:objet):vide;
fonction supprimerFilsGauche(ref S:sommet):vide;
fonction supprimerFilsDroit(ref S:sommet):vide;
fonction detruireSommet(ref S:sommet):vide;
fonction créerArbreBinaire(val Racine:objet):sommet;

**Arbres planaires**

fonction valeur(val S:sommetArbrePlanaire):objet;
fonction premierFils(val S:sommetArbrePlanaire):sommetArbrePlanaire;
fonction frere(val S:sommetArbrePlanaire):sommetArbrePlanaire;
fonction pere(val S:sommetArbrePlanaire):sommetArbrePlanaire;
fonction créerArborescence(val racine:objet):sommetArbrePlanaire;
fonction ajouterFils(ref S:sommetArbrePlanaire,val x:objet):vide;
fonction supprimerSommet(ref S:sommetArbrePlanaire):vide;

**Arbres binaire de recherche**

fonction ajouter(ref A:arbreBinaire d'objets, val v:objet):vide;
fonction supprimer(val A: arbreBinaire d'objets, val v:objet):vide;

**Tas**

fonction valeur(ref T:tas d'objet): objet;
fonction ajouter(ref T:tas de objet, val v:objet):vide;
fonction supprimer(val T:tas de objet):vide;
fonction creerTas(ref T:tas,val:v:objet):vide;
fonction detruireTas(ref T:tas):vide;

**File de priorité**

fonction changeValeur(ref T:tas d'objet,val s:sommet,val v:objet):vide;

**Dictionnaire**

fonction appartient(ref d:dictionnaire,val M::mot):booléen;
fonction creerDictionnaire(ref d: dictionnaire):vide ;
fonction ajouter(ref d:dictionnaire,val M::mot):vide;
fonction supprimer(ref d:dictionnaire,val M:mot):vide;
fonction detruireDictionnaire(ref d:dictionnaire):vide;

**Table de Hachage**

fonction chercher(ref T:tableHash de clés, val v:clé):curseur;
fonction créerTableHachage(ref T: tableHash de clé, ref h:fonction):vide;
fonction ajouter(ref T:tableHash de clé,val x:clé):booleen;
fonction supprimer((ref T:tableHash de clé,val x:clé):vide;

*FIN*